Genetic Algorithm

Application : Mathematical Function Optimization

```
import numpy as np
import random

# Define the problem: The function to optimize
def fitness_function(x):
    return x * np.sin(x)

# Generate the initial population
def create_population(size, x_min, x_max):
    return np.random.uniform(x_min, x_max, size)

# Evaluate fitness for the entire population
def evaluate_fitness(population):
    return np.array([fitness_function(ind) for ind in population])

# Selection: Roulette wheel selection
def select_parents(population, fitness):
    fitness = fitness - np.min(fitness) + 1e-6  # Shift fitness values to be positive
    total_fitness = np.sum(fitness)
    probabilities = fitness / total_fitness  # Normalize to sum to 1
    return population[np.random.choice(len(population), size=2, p=probabilities)]

# Crossover: Single-point crossover
def crossover(parent1, parent2, crossover_rate):
    if random.random() < crossover_rate:
        point = random.randint(0, 1)  # Single-point crossover for simplicity
        return (parent1, parent2) if point == 0 else (parent2, parent1)
    return parent1, parent2

# Mutation: Apply random changes
def mutate(individual, mutation_rate, x_min, x_max):
    if random.random() < mutation_rate:
        mutation_value = np.random.uniform(-1, 1)
        individual += mutation_value
```

```
        individual = np.clip(individual, x_min, x_max)  # Ensure within bounds
    return individual

# Main Genetic Algorithm
def genetic_algorithm(population_size, mutation_rate, crossover_rate, num_generations, x_min,
x_max):
    population = create_population(population_size, x_min, x_max)
    best_solution = None
    best_fitness = -np.inf

    for generation in range(num_generations):
        fitness = evaluate_fitness(population)

        # Track the best solution
        max_fitness_index = np.argmax(fitness)
        if fitness[max_fitness_index] > best_fitness:
            best_fitness = fitness[max_fitness_index]
            best_solution = population[max_fitness_index]

        new_population = []
        for _ in range(population_size // 2):  # Produce new population
            parent1, parent2 = select_parents(population, fitness)
            offspring1, offspring2 = crossover(parent1, parent2, crossover_rate)
            offspring1 = mutate(offspring1, mutation_rate, x_min, x_max)
            offspring2 = mutate(offspring2, mutation_rate, x_min, x_max)
            new_population.extend([offspring1, offspring2])

        population = np.array(new_population)

    return best_solution, best_fitness

# Take Genetic Algorithm parameters as inputs
population_size = int(input("Enter the population size: "))
mutation_rate = float(input("Enter the mutation rate (0 to 1): "))
crossover_rate = float(input("Enter the crossover rate (0 to 1): "))
num_generations = int(input("Enter the number of generations: "))
x_min = float(input("Enter the minimum value of x: "))
x_max = float(input("Enter the maximum value of x: "))
```

```
# Run the Genetic Algorithm
best_solution, best_fitness = genetic_algorithm(population_size, mutation_rate, crossover_rate,
num_generations, x_min, x_max)
print(f"Best Solution: x = {best_solution}")
print(f"Best Fitness: f(x) = {best_fitness}")
```

Output :

```
Enter the population size: 10
Enter the mutation rate (0 to 1): 0.10
Enter the crossover rate (0 to 1): 0.8
Enter the number of generations: 50
Enter the minimum value of x: 0
Enter the maximum value of x: 10
Best Solution: x = 8.208916912223948
Best Fitness: f(x) = 7.697246776822652
```